Back-end Automated Code Review (Sprint 3)

Introduction

Purpose

The purpose of this document is to outline our code review process which plays a crucial role in maintaining high code quality and identifying potential issues early. Code reviews help ensure that our software is robust, scalable, and secure.

Scope

This documentation covers the automated code review processes for the SciGallery project, focusing on backend services and critical frontend components.

Code Review Process

Participants

- **YUN DAI -** Backend developer (Code Submitter)
- **Qingyuan Yang** - Backend developer
- **AI Reviewer** - ChatGPT 4o

Frequency

Code reviews are conducted bi-weekly before each sprint ends, or after significant commits to our Git repository, such as new feature additions or critical bug fixes.

Selection Criteria

Criteria for Code Selection

Code is selected for review based on several factors including:

- Complexity of the code.
- The criticality of the application area affected.
- Recentness of changes.
- Modules with a history of frequent issues.

Tools and Techniques Used

ChatGPT 4o

Review Outcomes

Number of Reviews Conducted

In the last quarter, we conducted 10 code reviews.

Issues Identified

A total of 6 issues were identified.

Example of one of the code review:

Code:

```java
@Service  uziiiiiiiiiiiii
public class SchoolServiceImpl implements SchoolService {

    @Autowired
    private SchoolMapper schoolMapper;



    @Override  1 usage  uziiiiiiiiiiiii
    public PageBean page(Integer page, String name) {
        PageHelper.startPage(page,  pageSize: 10);

        List<School> schoolList = schoolMapper.list(name);
        Page<School> p = (Page<School>) schoolList;

        PageBean pageBean = new PageBean(p.getTotal(), p.getResult());
        return pageBean;
    }

    @Override  1 usage  uziiiiiiiiiiiii
    public void delete(Integer id) {
        schoolMapper.deleteById(id);
    }

    @Override  uziiiiiiiiiiiii
    public void add(School school) {
        schoolMapper.insert(school);
    }

    @Override  1 usage  uziiiiiiiiiiiii
    public School getById(Integer id) {
        return schoolMapper.getById(id);
    }

    @Override  uziiiiiiiiiiiii
    public void update(School school) {
        schoolMapper.update(school);
    }
}
```

ChatGPT Review:

Documentation Defects:

1. **Naming:**
   - The names of the methods (`page`, `delete`, `add`, `getById`, `update`) are concise and descriptive.
   - The variable names (`schoolMapper`, `schoolList`, `page`, `name`, `p`, `pageBean`) are clear and indicate their purpose.
2. **Comment:**
   - The code lacks comments. Adding comments to describe the purpose of each method and critical sections within methods would improve readability and maintainability.

Visual Representation Defects:

1. **Bracket Usage:**
   - Bracket usage is correct and consistent throughout the code.
2. **Indentation:**
   - Indentation is consistent and adheres to standard Java conventions.
3. **Long Line:**
   - There are no excessively long lines that hinder readability.

Structure Defects:

1. **Dead Code:**
   - There is no dead code present in the provided snippet.
2. **Duplication:**
   - There is no significant code duplication. Each method performs a unique function.

New Functionality:

1. **Use Standard Method:**
   - The code utilizes standard methods provided by the Spring framework and the PageHelper library.

Resource Defects:

1. **Variable Initialization:**
   - All variables are properly initialized before use.
2. **Memory Management:**
   - Memory management is handled appropriately. The use of `PageHelper` for pagination helps manage memory by limiting the amount of data retrieved at once.

Check Defects:

1. **Check User Input:**
   - There is no validation for user input (e.g., `page` and `id` parameters). Adding input validation would enhance the robustness of the code.

Interface Defects:

1. **Parameter:**
   - All methods have the correct parameters for their respective functions.

Logic Defects:

1. **Compute:**
   - The logic seems correct, but input validation and error handling should be considered.
   - The `page` method could be enhanced by checking if `schoolList` is not empty before casting it to `Page<School>` to avoid potential casting errors.
2. **Performance:**
   - The code is efficient in terms of performance. The use of `PageHelper` optimizes the database query by paginating the results.

Summary of Recommended Changes:

1. **Add Comments:**
   - Add comments to describe the purpose of each method and critical sections within the methods.

Summary of Recommended Changes:

1. **Add Comments:**
   - Add comments to describe the purpose of each method and critical sections within the methods.
2. **Input Validation:**
   - Add validation for user inputs (e.g., checking if `page` and `id` are non-negative and valid).
3. **Error Handling:**
   - Implement error handling to manage potential exceptions, such as database access issues or invalid input values.
4. **Check for Empty List:**
   - Add a check to ensure `schoolList` is not empty before casting it to `Page<School>` in the `page` method.

Feedback and Actions

Review of AI Feedback

Feedback from ChatGPT was reviewed by the development team within one day of the automated review completion. Priority was given to addressing critical vulnerabilities first.

Actions Taken After Review

Changes Implemented

- **Add Comments:** Enhance the code with comments explaining each major step.
- **Refactor Duplication:** Extract the repeated logic for error handling into a separate method to reduce duplication and improve code manageability.
- **Code smells:** Refactoring was performed to improve code readability and maintainability.

Unchanged Code and Reasons

- During our recent code review, it was noted that there are updated solutions available for some of the technologies and methodologies we are currently using. However, after considering the time required to adopt these new solutions, we have decided to maintain our existing technology stack.

Evaluation and Continuous Improvement

Effectiveness of Reviews

The reviews significantly reduced the number of runtime errors and improved the security posture of the application.

Lessons Learned and Improvements

We learned the importance of immediate action on critical issues and plan to integrate more detailed checklists for reviewers to cover edge cases better in future reviews.

Conclusion

The automated code review process has been instrumental in maintaining the quality and security of our software. Ongoing improvements and adaptations will continue to refine our review processes.